GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, W. Banzhaf, et al. (eds.)

# Size Fair and Homologous Tree Genetic Programming Crossovers

**W. B. Langdon**
Centrum voor Wiskunde en Informatica, Kruislaan 413, NL-1098 SJ Amsterdam
W.B.Langdon@cwi.nl http://www.cwi.nl/~bill/ Tel: +31 20 592 4093, Fax: +31 20 592 4199

## Abstract

Size fair and homologous crossover genetic operators for tree based genetic programming are described and tested. Both produce considerably reduced increases in program size and no detrimental effect on GP performance. GP search spaces are partitioned by the ridge in the number of program v. their size and depth. A ramped uniform random initialisation is described which straddles the ridge. With subtree crossover trees increase about one level per generation leading to sub-quadratic bloat in length.

## 1 INTRODUCTION

It has been known for some time that programs within GP populations tend to rapidly increase in size as the population evolves. If unchecked this consumes excessive machine resources. This is usually addressed either by enforcing a size or depth limit on the programs or by an explicit size penalty in the fitness measure, although other techniques may be used. Both main approaches have problems.

It has been shown that the protective effect of inviable code (which does not effect the fitness of the program) is not sufficient to explain all cases of bloat. Indeed there are at least two mechanisms involved [8]. However we also suggest these are manifestations of an underlying cause: *any stochastic search technique, such as GP, will tend to find the most common programs in the search space of the current best fitness.* Since in general there are more of these which are long than there are which are short (but GP starts with the shorter ones) the population tends to be filled with longer and longer programs. The exponential growth in the number of programs with size is a very strong driving factor. It may be the cause of bloat even if the fitness function changes rapidly or we penalise programs with the same fitness as their parents [7].

Using this argument we devised an unbiased tree mutation operator which carefully controls variation in size and produces much less bloat [8]. In Section 2 we describe the corresponding crossover operator and in Section 3 we describe means of extending it to increase the chance of crossover between like parts of parent trees yielding a more homologous operator. We compare the evolution of tree size and depth for the three crossover operators starting from three types of initial random populations: standard "ramped half-and-half" [4, pages 92–93] "ramped half-and-half" with bigger initial trees and ramped uniform random (described in Section 4). In Section 5 we compare both new operators with standard subtree crossover on two continuous domain problems (symbolic regression of the quintic and sextic polynomials) and two discrete problems (Boolean 6 multiplexor and 11 multiplexor). This is followed by a discussion in Section 6 and we conclude in Section 7.

## 2 SIZE FAIR CROSSOVER

In size fair crossover we select in the normal way two parents and the crossover point in the first parent, i.e. the one which supplies the root node. The difference is the choice of the second crossover point. The size of the subtree to be deleted from the first parent is calculated and this is used to guide the random choice of the second crossover point. The size of every subtree in the second parent is calculated. Those bigger than $1{+}2 \times |\text{subtree to be deleted}|$ are excluded. Note each offspring will be no more than $|\text{subtree to be deleted}| + 1$ nodes longer than its first parent. For the remainder, we count the number that are shorter ($n_-$), the same ($n_0$) and longer ($n_+$) than the subtree to be deleted. We also calculate the mean size difference for both smaller ($mean_-$) and bigger ($mean_+$) subtrees. If there are no smaller or no bigger trees then the size of the inserted subtree will be equal to that of the subtree to be deleted. Note a terminal is always replaced by another terminal.

We use a biased roulette wheel to select the length of the subtree. If there is more than one subtree of the desired length, we choose between them uniformly at random. Thus the chance of a subtree being selected falls in proportion to the number of other subtrees in the second parent of the same size. The probability fair crossover make no change in size is $p_0 = 1/|\text{subtree to be deleted}|$. All the shorter lengths have the same probability of being selected, as do all the longer lengths. However we use the mean size difference to balance these two probabilities so that on average there is no change of length. I.e. $p_+ = \frac{1-p_0}{n_+(1+mean_+/mean_-)}$

## 3 HOMOLOGOUS CROSSOVER

Standard GP crossover moves code fragments from one program to another. It is assumed that since the code fragment has survived the selection process, it must have some worth and so using it to create a new program is more likely to produce a better program. However it can be anticipated that the worth of a code fragment will depend upon the context within which it is executed. Moving into a different program at a random location may destroy this context [10]. Secondly the presence of bloat may indicate that the code fragment is not good, only that has survived the selection process by being not harmful. With this in mind several context preserving crossovers have been suggested [2, 11] (and [9] for linear GP). These aim to increase the chance of moving the code fragment to a (syntactically) similar part of the recipient program and thus preserve its context and so worth. Some of these have only had mixed success and so we propose a new homologous crossover operator.

The homologous crossover operator works identically to the size fair crossover operator up to the last step. Instead of randomly choosing between all the available subtrees in the second parent of the desired size in homologous crossover we deterministically choose the one closest to the subtree in the first parent. We define the distance between the two crossover points using only their locations and the shapes of the two trees. The closeness of two points within the trees is given by the depth at which their routes back to the root diverge. Note homologous crossover on two identical trees will produce an identical offspring if the offspring is of the same size. On average in quintic polynomial runs with standard initial populations 16% of homologous crossovers produced an offspring identical to its first parent. In contrast only 7% of fair and 5% standard crossovers did.

## 4 RAMPED UNIFORM INITIALISE

In [14] binary tree populations are shown evolving away from both full or sparse trees. In fact towards the most common tree shape [8]. In this section we describe a new means of creating the initial population in which the population starts with common trees of a range of lengths. We anticipate that such a population will evolve to bigger trees but remain near the most common tree shape (for a given length).

Uniform sampling as described by Iba [3] not only ensures almost all the initial population has one of the common shapes but also ensures they are near the maximum possible length. Instead we adopt a more gradualist approach similar to "ramped half-and-half" and instead generate a uniform range of program sizes. Our algorithm is similar to Iba's but is fast and stable even for large trees. The algorithm to generate a random tree of a given length is based upon Alonso's bijective algorithm [1]. C++ code can be found at `ftp.cs.bham.ac.uk /pub/authors/W.B.Langdon/gp-code/rand_tree.cc`.

Firstly a program length is chosen at random. Next one of the combinations of numbers of function arities which can create trees of this size is chosen at random. Since each combination of arities corresponds to a different number of programs, the choice is biased in proportion to this number. Then a random tree is created and converted to a random program by labeling its internal nodes with functions of the same arity chosen at random from the function set and its leafs are labeled with random terminals. (In the symbolic regression problems the chance of choosing $x$ is 50%). Ramped uniform produces programs with shapes near the ridge in the search space, while "ramped half-and-half" produces many more large full trees.

## 5 EXPERIMENTS

The four bench mark problems are symbolic regression of the quintic polynomial [5] symbolic regression of the sextic polynomial [5] learning the Boolean 6-multiplexer [4, page 187] and the Boolean 11-multiplexer functions [4]. Apart from the absence of size or depth restrictions and the use of tournament selection our GP runs are essentially the same as [4] and [5]. Parameters are summarised in Tables 1 and 2

Table 1: GP Parameters for the Symbolic Regression

| | |
|---|---|
| Objective | quintic polynomial $x^5 - 2x^3 + x$ (sextic polynomial $x^6 - 2x^4 + x^2$) |
| Terminals | $x$ and 250 random constants |
| Functions | $+ - \times$ % (protected division) |
| Fitness | Mean error in 50 fitness cases |
| Hits | Number fitness cases where error $< 0.01$ |
| Selection | Tournament 7, non-elitist, generational |
| Pop Size | 4000 |
| Parameters | 90% one child crossover, no mutation. |
| Termination | Maximum number of generations 50 |

Table 2: GP Parameters for Multiplexor Problems

| | |
|---|---|
| Objective | Evolve the Boolean 6 (11) multiplexor |
| Terminals | D0 D1 D2 D3 (D4 D5 D6 D7) A0 A1 (A2) |
| Functions | AND OR IF NOT |
| Fitness | The $2^6$ ($2^{11}$) combinations of the inputs |
| Pop size | 500 (4000) |

(Multiplexor is as Table 1 unless stated). We speed up GP on the two Boolean problems by extending the bit packing technique described in [12] to IF. This enabled us to evaluate 32 fitness cases simultaneously.

To test the importance of the initial population we carried out experiments with both the standard "ramped half-and-half" method and also using it to create bigger trees with maximum depths between 5 and 8, corresponding to binary (multiplexor) trees up to a length of 255 (3280 in principle although the maximum observed was 611). Duplicate prevention was not used. The range of random program sizes created using the ramped uniform method was chosen to have the same minimum size and similar mean size to standard "ramped half-and-half". (Note "ramped half-and-half" produces a small fraction of very big trees; much bigger than the biggest we created using ramped uniform).

For each of the four problems we performed 50 runs for each combination of crossover type and means of creating the initial population. The results of these $4 \times 50 \times 3 \times 3$ runs are summarised in Table 3.

Table 3: Means of 50 runs with each crossover

| Experiment | | | num sol | effort 000 | Sol size mean | Sol size min–max | Final pop mean | Final pop max |
|---|---|---|---|---|---|---|---|---|
| Qu | R2–6 | stand | 39 | 660 | 218 | 15–1205 | 752 | 3276 |
| | R2–6 | fair | 38 | 630 | 63 | 15– 153 | 116 | 251 |
| | R2–6 | homo | 37 | 670 | 61 | 17– 157 | 85 | 162 |
| Qu | R5–8 | stand | 29 | 1000 | 352 | 27–1871 | 815 | 3169 |
| | R5–8 | fair | 32 | 880 | 106 | 27– 337 | 147 | 277 |
| | R5–8 | homo | 29 | 970 | 77 | 25– 177 | 113 | 213 |
| Qu | U3–25 | stand | 42 | 520 | 337 | 15–1485 | 1188 | 5124 |
| | U3–25 | fair | 39 | 610 | 60 | 15– 145 | 157 | 381 |
| | U3–25 | homo | 28 | 950 | 50 | 17– 119 | 147 | 354 |
| Sex | R2–6 | stand | 13 | 3100 | 451 | 53–1209 | 735 | 2852 |
| | R2–6 | fair | 7 | 4400 | 75 | 15– 139 | 119 | 251 |
| | R2–6 | homo | 9 | 3900 | 61 | 15– 177 | 105 | 209 |
| Sex | R5–8 | stand | 32 | 920 | 408 | 31–1019 | 919 | 3415 |
| | R5–8 | fair | 26 | 1300 | 116 | 29– 321 | 164 | 307 |
| | R5–8 | homo | 22 | 1300 | 88 | 27– 181 | 122 | 219 |
| Sex | U3–25 | stand | 26 | 1300 | 633 | 61–2037 | 1332 | 5446 |
| | U3–25 | fair | 25 | 1300 | 123 | 35– 235 | 190 | 408 |
| | U3–25 | homo | 19 | 1900 | 107 | 15– 205 | 171 | 360 |
| 6M | R2–6 | stand | 39 | 38 | 96 | 15– 275 | 731 | 2573 |
| | R2–6 | fair | 47 | 24 | 47 | 10– 160 | 138 | 260 |
| | R2–6 | homo | 46 | 32 | 42 | 10– 114 | 121 | 236 |
| 6M | R5–8 | stand | 45 | 42 | 205 | 34– 845 | 852 | 2734 |
| | R5–8 | fair | 47 | 30 | 118 | 36– 324 | 206 | 349 |
| | R5–8 | homo | 45 | 44 | 110 | 28– 266 | 177 | 308 |
| 6M | U3–25 | stand | 33 | 36 | 59 | 12– 435 | 655 | 2781 |
| | U3–25 | fair | 26 | 64 | 36 | 14– 104 | 133 | 283 |
| | U3–25 | homo | 24 | 75 | 35 | 10– 189 | 128 | 277 |
| 11M | R2–6 | stand | 37 | 750 | 292 | 57–1344 | 684 | 2832 |
| | R2–6 | fair | 49 | 270 | 93 | 35– 228 | 176 | 368 |
| | R2–6 | homo | 47 | 290 | 79 | 25– 207 | 156 | 338 |
| 11M | R5–8 | stand | 10 | 4100 | 439 | 223– 894 | 679 | 2349 |
| | R5–8 | fair | 43 | 540 | 212 | 83– 452 | 248 | 481 |
| | R5–8 | homo | 32 | 960 | 221 | 77– 504 | 244 | 463 |
| 11M | U3–25 | stand | 36 | 680 | 251 | 90– 896 | 686 | 3172 |
| | U3–25 | fair | 18 | 1400 | 86 | 50– 116 | 179 | 399 |
| | U3–25 | homo | 24 | 930 | 86 | 53– 142 | 173 | 390 |

### 5.1 EVOLUTION OF SIZE

In all 36 cases we see the GP population bloats. (The initial populations start with mean sizes near 14, or 75 for R 5–8). However there is a clear separation between standard crossover and the two new crossovers. In all cases standard crossover produces far bigger trees. (The mean length of programs at the end of the runs is given in column 9 of Table 3. While the last column gives the average size of the biggest program at the end of the run). This is also reflected in the fact that it also produces bigger solutions. There isn't such a clear cut difference between fair and homologous crossover.

Figure 1 shows the evolution of program lengths in the population for the quintic symbolic regression problem starting from R 2–6 initial populations. It shows the typical behaviour, where program size and the spread of sizes in the population in runs using standard crossover grow rapidly and non-linearly. In contrast

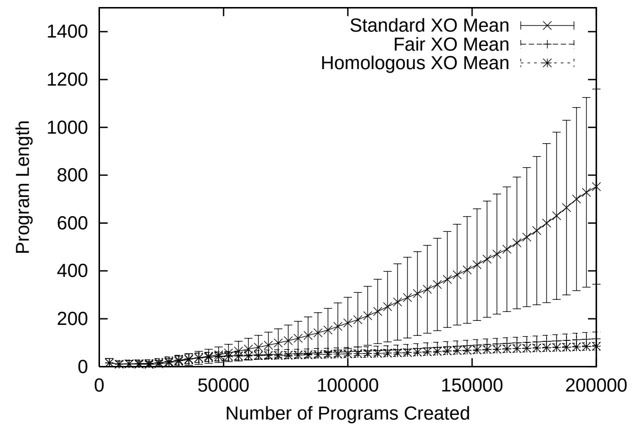


Figure 1: Evolution of population program length from R 2–6 populations. Error bars indicate standard deviation in population. Mean of 50 runs of quintic polynomial problem.

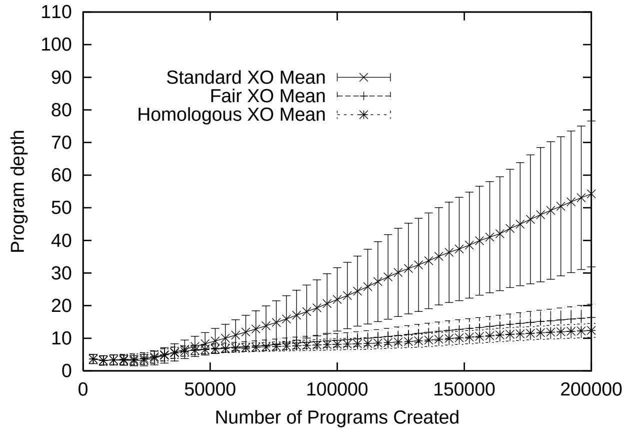


Figure 2: Evolution of population program depth. Error bars indicate standard deviation in population. Means of 50 quintic polynomial runs.

both fair crossover and homologous crossover show the hoped for reduction in bloat. In both these cases growth in program size is much slower and more linear.

## 5.2 EVOLUTION OF DEPTH

Figure 2 shows the evolution of program depths for the same quintic runs as Figure 1. It shows the typical behaviour, where both program depth and the spread of depths in the population in runs using standard crossover grow rapidly but apparently linearly. Over the last 3/4 of the run the mean growth is 1.2 layers per generation. Which greatly exceeds 0.2 for fair and homologous crossover runs.

Figure 3 shows the evolution of program depths for each our four problems and each of the three methods of creating the initial population. It is evident that the linear growth in program mean depth is not a fluke but may be an important property of standard subtree crossover (in the absence of depth or size limits). Table 4 gives the mean and max program depths and their average rate of increase over the last 38 generations of the runs. While not problem independent, Table 4 shows the rate of increase in depth is consistently close to unity. As discussed in [8] this, together with remaining close to the ridge in the number of programs versus their shape leads to a prediction of growth of sub-quadratic growth in program length (for modest size programs we expect size $O(\text{gens}^{1.3})$ rising to a limit of quadratic growth for $|\text{program}| \gg 1000$. Over the last 38 generations the measured values are near $O(\text{gens}^{1.25})$).

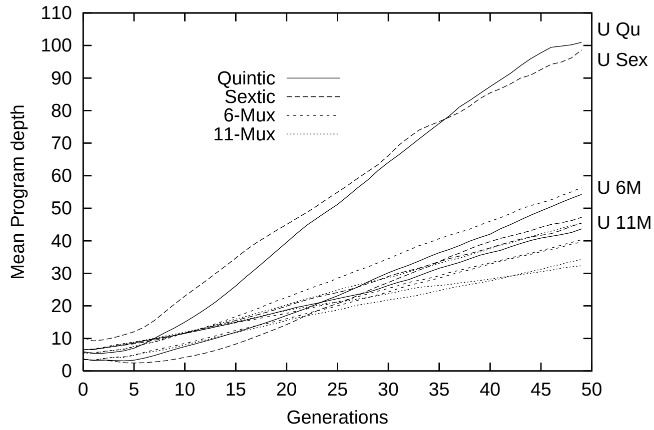


Figure 3: Evolution of population program depth. Means of 50 runs with standard crossover for each problem and initial populations.

Table 4: Program Depth, standard crossover 50 runs

| Problem | Initialisation | Final pop mean | Final pop max | Growth per gen mean | Growth per gen max |
|---|---|---|---|---|---|
| Quintic | R2–6 | 54 | 181 | 1.2 | 4.0 |
| | R5–8 | 43 | 128 | 0.8 | 2.4 |
| | U 3–25 | 101 | 332 | 2.2 | 7.0 |
| Sextic | R2–6 | 47 | 150 | 1.1 | 3.5 |
| | R5–8 | 45 | 131 | 0.9 | 2.5 |
| | U 3–25 | 98 | 312 | 1.9 | 5.8 |
| 6-Mux | R2–6 | 39 | 101 | 0.8 | 2.1 |
| | R5–8 | 40 | 97 | 0.7 | 1.9 |
| | U 3–25 | 56 | 172 | 1.2 | 3.6 |
| 11-Mux | R2–6 | 34 | 107 | 0.7 | 2.1 |
| | R5–8 | 32 | 90 | 0.5 | 1.4 |
| | U 3–25 | 45 | 157 | 0.9 | 2.9 |

### 5.3 EVOLUTION OF SHAPE

Figure 4 shows the average evolution of all 450 initial populations used in the quintic polynomial problem (the behaviour of the other three problems is similar). For standard crossover (+) Figure 4 shows GP population behave much as they do for other problems [8], with programs tending both to grow bigger and deeper but also tending to be near the combination of size and depth for which there are most programs.

We see both fair (×) and homologous (□) crossover producing trees of similar shapes as standard crossover (+) (again near the peak number of programs) but moving much more slowly along a similar trajectory.

As shown in [8] for very different problems standard crossover evolves the population towards the peak in the distribution of programs versus their shape. However like [14] the population retains a long term memory of how it was initialised and the mean evolutionary curves do not coalesce. This is consistent with the view that on average populations follow the steepest gradient in the density of programs. Apart from nearly full trees the gradient is almost parallel to the y-axis with only a little component towards the ridge and so steepest ascent routes do not rapidly coalesce on the ridge. (Note the population mean in individual runs wonders considerably either side of the ridge).

Again we see fair and homologous runs show much reduced bloat (the tick marks every five generations are much closer together) and lie close to each other. However both runs with bigger initial populations and those produced by ramped uniform deviate from the mean shape followed by standard crossover runs and create deeper trees. This may be because, while size change is carefully controlled, no specific restrictions are placed on depth exploration, allowing the population to move more freely in this direction. Future genetic operators might consider this aspect of bloat.

### 5.4 SEARCH EFFICIENCY

As shown in Table 3 in all four problems most of the nine experiments have similar search efficiency in terms of number of solutions found or "effort" [4, page 194]. Even with 50 runs, there are two cases where the difference can be thought statistically reliable, even though in others the differences may be large. 1) all three crossover operators perform slightly better with the two new means of creating the initial random programs in the sextic polynomial and 2) in the 11-multiplexor problem standard crossover performs slightly worse on large initial programs. I.e. the new operators perform at least as well as the original.

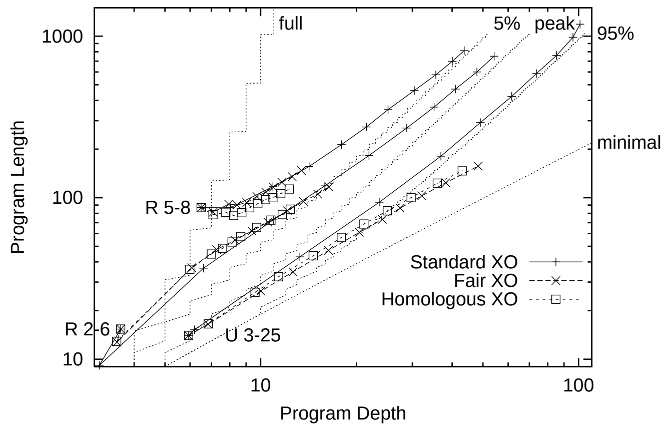


Figure 4: Evolution of mean population program shape showing effect of three types of initial populations. Tick marks every 5 generations. The full tree and minimal tree limits are shown with dotted lines, as are the most likely shape (peak) and the 5% and 95% limits (which enclose 90% of all programs of a given size). Means of 50 of quintic polynomial runs. Note log log scales.

### 5.5 HOMOLOGOUS MEASUREMENTS

It is disappointing that homologous crossover shows little performance gain over fair crossover. We expect it to increase the convergence of the GP populations. In particular, in the multiplexor runs, we would hope to see common trees evolving with combinations of address bits as the first arguments of IF functions and data bits as the second and third arguments. Using population variety and number of duplicate children produced we do see a little evidence for some convergence but it doesn't appear to have a big impact on the spread of fitness values or search efficiency.

## 6 DISCUSSION

The impressive suppression of bloat produced by fair crossover was expected as it concurs with our theory of bloat [8] and similar results for fair mutation. While they are both designed with a view to reducing bloat by carefully controlling how the search space is sampled (i.e. by sampling programs of neighbouring lengths) and alternative view of their success, is by closely correlating the size of the inserted subtree with that of the removed they suppress the "removal bias" [15] bloat mechanism and remaining bloat is due to some other mechanism probably inviable code [6]. It is also possible that reduced rate of growth derives from the upper bound on the size of the replacement subtree in both cases.

The simple linear growth in mean depth of near one level per generation gives a simple problem independent prediction of when a population will be severely affected by a depth limit. The curve indicating the ridge in the distribution of programs against their size and shape is known for binary trees [13] and can be precalculated for more complex function sets. Thus given a predicted depth this may be converted into a predicted program size. We predict that standard GP will run into common depth (17 layers) or size limit (which can be as low as 50 or 200 nodes), within a few generations and certainly before the 50 generations commonly used.

## 7 CONCLUSIONS

We have presented and demonstrated on four bench mark problems a new bloat reduced crossover operator, a new homologous crossover operator and a new mechanism for creating random populations for tree based genetic programming. The results in terms of reduction in growth of both mean and maximum program and solution sizes are impressive and are achieved with out reduction in search efficiency.

While we have demonstrated the homologous crossover operator is effective at finding solutions and reducing bloat, we have not yet shown it to be greatly more efficient. Growth in program sizes was found not to depend overly on the initial population however it does have a dominant role in the evolution of program shapes. The ridge in the distribution of programs for each size and shape acts to divide the search space. "Ramped half-and-half" does not search a large part of the search space corresponding to long thin trees (and vice-versa an initial population of long thin trees may not search the part of the search space corresponding to short bushy trees).

Average growth in program depth when using standard subtree crossover is near linear in these problems. When combined with the known distribution of programs, this yields a prediction of sub-quadratic growth in program size. This indicates GP populations using standard crossover (and no parsimony techniques) will quickly reach bounds on size or depth commonly used.

## References


[1] L. Alonso and R. Schott. *Random Generation of Trees.* Kluwer Academic Publishers, 1995.

[2] P. D'haeseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, 27-29 June 1994. IEEE Press.

[3] H. Iba. Random tree generation for genetic programming. In H.-M. Voigt *et. al.* editors, *Parallel Problem Solving from Nature IV*, 1996. Springer.

[4] J. R. Koza. *Genetic Programming.* 1992.

[5] J. R. Koza. *Genetic Programming II.* 1994.

[6] W. B. Langdon. The evolution of size in variable length representations. In *1998 IEEE International Conference on Evolutionary Computation.*

[7] W. B. Langdon and R. Poli. Genetic programming bloat with dynamic fitness. In W. Banzhaf *et. al. Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS*, 1998. Springer-Verlag.

[8] W. B. Langdon, T. Soule, R. Poli, and J. A. Foster. The evolution of size and shape. In L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 8, MIT Press, 1999.

[9] P. Nordin, W. Banzhaf, and F. D. Francone. Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. In L. Spector *et. al.* editors, *Advances in Genetic Programming 3*, chapter 12, 1999.

[10] U.-M. O'Reilly and F. Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3* 1995. Morgan Kaufmann.

[11] R. Poli and W. B. Langdon. Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, 6(3):231–252, 1998.

[12] R. Poli and W. B. Langdon. Sub-machine-code genetic programming. In L. Spector *et. al.* editors, *Advances in Genetic Programming 3*, chapter 13.

[13] R. Sedgewick and P. Flajolet. *An Intoduction to the Analysis of Algorithms.* Addison-Wesley, 1996.

[14] T. Soule. *Code Growth in Genetic Programming.* PhD thesis, University of Idaho, Moscow, Idaho, USA, 15 May 1998.

[15] T. Soule and J. A. Foster. Removal bias: a new cause of code growth in tree based evolutionary programming. In *1998 IEEE International Conference on Evolutionary Computation* 1998.